**一、简介**

1、Git：Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

2、GitHub简介：gitHub是一个面向开源及私有[软件](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6/12053)项目的托管平台，是分布式版本控制系统，因为只支持git 作为唯一的版本库格式进行托管，故名gitHub。

3、参考网址

<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/>

**二、集中式版本控制和分布式版本控制区别**

1、集中式版本控制

版本库是集中存放在中央服务器的，而开发的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开发，开发结束之后再把自己的代码推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了再放回图书馆。集中式版本控制系统最大的弊端就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟。

 

集中式版本控制

分布式版本控制

2、分布式版本控制

分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

3、对比

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法开发了。

在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样开发，只是交换修改不方便而已。

1. **安装git**

1、Windows: <https://git-for-windows.github.io/>

2、Unbuntu Linux: 使用sudo apt-get install git命令下载安装，若下载不成功，可使用sudo apt-get install git-core命令下载安装

**四、指定本地名称**

1、Windows下可安装TortoiseGit（Git的可视化操作工具），并将TortoiseGit关联远程github。鼠标右键TortoiseGit-》setting，设置git的用户名和邮箱地址

2、使用命令：

$ git config --global user.name "Your Name"

$ git config --global user.email [email@example.com](mailto:email@example.com)

**五、创建版本库**

1、在本地新建一个空白目录，使用命令git init把这个目录变成了Git可以管理的仓库。则会在此目录下看到一个.git的目录，如果没有此目录，可以用ls –ah命令即可。

2、在此目录下新建一个文件

3、用命令git add告诉git，把文件添加到仓库

git add readme.txt（readme.txt为文件名称和类型）

git add可以添加多个文件，git add file2.txt file3.txt

4、用命令git commit告诉Git，把文件提交到仓库。-m后面输入的是本次提交的说明

git commit –m ”wrote a readme file”

**六、Git其他命令**

1、用命令git status查看仓库当前状态

2、用命令git diff查看文件哪里修改了

3、用命令git log命令显示从近到远提交的日志，如果看着很乱，可以加上—pretty=online参数

4、用命令git reset –hard commit\_id（commit\_id为提交的版本id）可以回退到任意版本

5、用命令git reflog查看命令历史

6、用命令git rm删除文件，并且git commit

7、git rm text.txt

8、用命令git checkout – text.txt从仓库中检出

9、用命令git pull 更新到最新版本

1. **远程仓库**

1、创建SSH Key 用命令

ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

一路回车即可。如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

2、登陆GitHub，打开“Account settings”，“SSH Keys”页面：



点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容。

点“Add Key”，你就应该看到已经添加的Key：

3、GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

**八、添加远程库**

1、登陆<https://github.com/>，点击Start a project，输入仓库的名称，仓库的描述，设置其权限，点击创建即可。（仓库名称为：gitTest）

2、将本地仓库与远程仓库关联

git remote add origin git@github.com:michaelliao/ gitTest.git

请千万注意，把上面的michaelliao替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

3、把本地库的所有内容推送到远程库上：

git push -u origin master

注意：SSH警告

当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告：

The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.

RSA key fingerprint is xx.xx.xx.xx.xx.

Are you sure you want to continue connecting (yes/no)?

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。

Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

Warning: Permanently added 'github.com' (RSA) to the list of known hosts.

这个警告只会出现一次，后面的操作就不会有任何警告了。

如果你实在担心有人冒充GitHub服务器，输入yes前可以对照[GitHub的RSA Key的指纹信息](https://help.github.com/articles/what-are-github-s-ssh-key-fingerprints/)是否与SSH连接给出的一致。

4、之后可以查看本地仓库和远程仓库里的内容是一模一样的。

以后只要本地做了提交，就可以通过命令git push origin master把本地master分支的最新修改推送到GitHub

**九、从远程仓库克隆**

1、上次我们讲了先有本地库，后有远程库的时候，如何关联远程库。现在，假设我们从零开发，那么最好的方式是先创建远程库，然后从远程库克隆。

2、首先，登陆GitHub，创建一个新的仓库，名字叫gitskills，我们勾选Initialize this repository with a README，这样GitHub会自动为我们创建一个README.md文件。创建完毕后，可以看到README.md文件，现在，远程库已经准备好了。

3、命令git clone克隆一个本地库：

git clone git@github.com:michaelliao/gitskills.git

注意把Git库的地址换成你自己的，然后进入gitskills目录看看，已经有README.md文件了

**十、分支管理**

1、分支概述

在[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)里，你已经知道，每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。

一开始的时候，master分支是一条线，Git用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点：



每次提交，master分支都会向前移动一步，这样，随着你不断提交，master分支的线也越来越长：

当我们创建新的分支，例如dev时，Git新建了一个指针叫dev，指向master相同的提交，再把HEAD指向dev，就表示当前分支在dev上：



你看，Git创建一个分支很快，因为除了增加一个dev指针，改改HEAD的指向，工作区的文件都没有任何变化！不过，从现在开始，对工作区的修改和提交就是针对dev分支了，比如新提交一次后，dev指针往前移动一步，而master指针不变：



假如我们在dev上的工作完成了，就可以把dev合并到master上。Git怎么合并呢？最简单的方法，就是直接把master指向dev的当前提交，就完成了合并：



所以Git合并分支也很快！就改改指针，工作区内容也不变！合并完分支后，甚至可以删除dev分支。删除dev分支就是把dev指针给删掉，删掉后，我们就剩下了一条master分支：



真是太神奇了，你看得出来有些提交是通过分支完成的吗？下面开始实战。2、新建分支并使用分支

首先，我们创建dev分支，然后切换到dev分支，之后通过命令git push origin dev将代码提交至远成仓库。

$ git checkout -b dev

Switched to a new branch 'dev'

$ git push origin dev

git checkout命令加上-b参数表示创建并切换，相当于以下两条命令：

$ git branch dev

$ git checkout dev

Switched to branch 'dev'

然后，用git branch命令查看当前分支：

$ git branch

\* dev

master

git branch命令会列出所有分支，当前分支前面会标一个\*号。然后，我们就可以在dev分支上正常提交，比如对readme.txt做个修改，加上一行：

Creating a new branch is quick.然后提交，之后使用命令git push origin dev将代码提交至远程仓库：

$ git add readme.txt

$ git commit -m "branch test"

[dev fec145a] branch test

1 file changed, 1 insertion(+)

之后可以在github中查看到提交的dev信息，现在，dev分支的工作完成。

3、切换分支

$ git checkout master

Switched to branch 'master'

切换回master分支后，再查看一个readme.txt文件，刚才添加的内容不见了！因为那个提交是在dev分支上，而master分支此刻的提交点并没有变：



3、合并分支

现在，我们把dev分支的工作成果合并到master分支上：

$ git merge dev

Updating d17efd8..fec145a

Fast-forward

readme.txt | 1 +

1 file changed, 1 insertion(+)

git merge命令用于合并指定分支到当前分支。合并后，再查看readme.txt的内容，就可以看到，和dev分支的最新提交是完全一样的。注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把master指向dev的当前提交，所以合并速度非常快。当然，也不是每次合并都能Fast-forward，我们后面会讲其他方式的合并。

4、删除分支

$ git branch -d dev

Deleted branch dev (was fec145a).

删除后，查看branch，就只剩下master分支了：

$ git branch

\* master

使用git push origin :dev命令删除远程仓库分支dev

git push origin :dev

To github.com:jhchuang1992/gitTest.git

- [deleted] dev

因为创建、合并和删除分支非常快，所以Git鼓励你使用分支完成某个任务，合并后再删掉分支，这和直接在master分支上工作效果是一样的，但过程更安全。